Release 19.8 Shared Variable Processor Version 2.0

Configuration, Operation and Maintenance Manual

SHARP APL/370

I.P. Sharp
A REUTER COMPANY

APL Systems Development Group
I.P. Sharp Associates Limited
2 First Canadian Place, Suite 1900
Toronto, Ontario, Canada
M5X 1E3

Telephone 416 364-5361
Telex 0622259 I P SHARP TOR

Preface

This document supercedes Chapter 1 of the SHARP APL/370 Auxiliary Processors Manual entitled 'SSVP; The SHARP Shared Variable Processor System' (Publication code: 0452 8703 E19).

This document is divided into a number of chapters. The first chapter is a brief overview of the SHARP Shared Variable Processor (SVP). This chapter emphasizes the differences between the Shared Variable Processor 2.0 and that of prior releases.

The next two chapters describe how the specifications for a Shared Variable Processor are satisfied. The DESIGN Chapter outlines all the major requirements of an SVP. The IMPLEMENTATION Chapter describes how these requirements are implemented, given the MVS and Sharp APL environments.

OPERATIONS outlines the operator interface to the Shared Variable Processor. This covers the MVS START, STOP, and MODIFY commands, obtaining and inspecting dumps, and console messages. WRITING AUXILIARY PROCESSORS: THE SVP INTERFACE describes the use of a set of macros provided as an interface to the SVP. Two methods for using these macros are discussed. CONFIGURATION provides a detailed description of the components of the SVP and the parameters of the SVP started task. INSTALLATION GUIDELINES is a list of suggestions that will optimize the performance of the SVP.

Prerequisites:

Release 19.8 of SHARP APL/370

Printed in Canada
March 1988
Publication Code: 0846 8803 e1

CONTENTS

INTRODUCTION

The SHARP APL Shared Variable Processor (SVP) provides communication between otherwise independent "tasks"; a task can be an MVS started task, batch job, or an APL user T-, N-, B-, or S-task. The SVP handles the storage and transfer of shared variable values, and provides a communications control protocol. A site may run one or more shared variable processors on a single host and each may be configured and accessed independently.

The Shared Variable Processor 2.0 is implemented as an MVS subsystem; there are two benefits of this. First, it uses cross-memory services in facilitating communication between tasks, in line with IBM recommendations. Second, as a started task, the SVP is visible to Operations staff, and may be manipulated with standard MVS system commands.

A programming interface to the Shared Variable Processor is provided. This is a set of macros and copy code written in 370 Assembly language. These routines enable a systems programmer to write an auxiliary processor that makes use of the SVP to communicate with other tasks.

THE SHARED VARIABLE PROCESSOR VERSION 1.0

Versions of the Shared Variable Processor prior to SVP 2.0 required an MVS Nucleus Generation during installation, as well as the modification of an MVS System Module (IEAVTRML). Standard trouble-shooting procedures, such as dumps, were only available through proprietary utilities. The Lockword mechanism used by the prior version to serialize access to SVP storage, occasionally caused timing errors.

THE SHARED VARIABLE PROCESSOR VERSION 2.0

Version 2.0 of the Shared Variable Processor overcomes a number of shortcomings of earlier releases. Shared Variable Processor 2.0 brings the operation of the SVP in line with standard MVS practices. The SVP is now an MVS subsystem, typically accessed via a type 3 SVC. Installation of the SVP now requires only easily maintained modifications to MVS parmlib documents, and an IPL. No modifications to any IBM Operating System code are required. All services used by the SVP are standard MVS facilities. The existence of an SVP address space allows an MVS lock to replace the prior Lockword mechanism. These aspects of the SVP also facilitate the use of standard MVS troubleshooting tools, installation, and maintenance.

Data communications, in general, refers to the transfer of data between two separate locations. In the mainframe computer environment, it is tasks which must communicate. Multi-tasking, virtual storage, and other factors produce a scenario in which concurrent tasks are separated logically, as well as geographically. There is a requirement for these tasks to initiate communication and exchange information. Stated simply, a Shared Variable Processor satisfies this requirement; communication between tasks.

In the context of Sharp APL running under MVS, there are a variety of requirements for communication. At a low level, an APL workspace must interface with peripherals such as terminals and printers. At a higher level, there is a requirement for users of the system to be able to share information and variables. Also, auxiliary processors provide tasks (APL and non-APL) access to Operating System data and services, and there must be some communication medium between these tasks and auxiliary processors. There are a wide variety of candidates for partnership in communication, then, and these vary in complexity.

A Shared Variable Processor is a task that must establish a bridge between two other tasks. It must allow two partners to be able to set the value of a single variable and to reference the value of that variable. It must also provide a communication protocol to ensure that partners are setting and referencing variables in a timely and meaningful fashion. Lastly, several sets of partners must be able to use the SVP concurrently.

DESIGN REQUIREMENTS OF A SHARED VARIABLE PROCESSOR

The Shared Variable Processor must be designed to relay a variety of information on behalf of the user. The most obvious information is the shared variable value. As well, the SVP must be able to pass along identification of a task to another task. Further, when a task terminates sharing of a variable or signs off the SVP, its partner must be informed.

A Shared Variable Processor should be self-contained and independent; the number and configuration of SVP's within a single host system should be a matter of discretion at the site.

A Shared Variable Processor should fit into the host operating system in standard fashion. The SVP should conform to standard operational and maintenance procedures.

The design of a Shared Variable Processor must take into account the central role it will play in many vital applications. Thus it must meet high standards of performance and reliability.

FUNCTIONAL REQUIREMENTS OF A SHARED VARIABLE PROCESSOR

The Shared Variable Processor must try to create the illusion that two partners are accessing and manipulating a single variable. Given the constraints of MVS, and the fact that the partners are discrete tasks, a perfect illusion is unachievable. There are up to three copies of a single shared variable; one that is visible to each partner, and one that is stored within the shared variable processor. The SVP must maintain all three copies.

When two partners are sharing a variable, there must be some mechanism in place that can be used to ensure that communication is synchronized. For example, user A may not want to set the value of a shared variable before user B has accessed the previous value.

Within Sharp APL, shared variable requirements fall into two categories; those that can be handled from within the the APL address space and those that require external resources. For economic reasons, the Shared Variable Processor must provide separate treatment for each type of request.

Lastly, there must be some provision for site-written Auxiliary Processors to use the Shared Variable Processor. A set of routines must be provided that allow an Auxiliary Processor to sign on and sign off the SVP and to check its status, to set up and terminate communication with another task, to inquire as to other tasks which may be trying to set up communication, to handle interlocking, and to set and reference variables.

MVS SUBSYSTEM STRUCTURE

There is no restriction on the number of SVPs that may run under a single MVS system. Each SVP runs as an MVS subsystem. An SVP operation is normally initiated with an SVC call. Some number of MVS site SVCs will be dedicated to SVP operations (often just one), and each will have a default SVP associated with it. Each of these SVCs requires a separate secondary or non-active subsystem. These SVC subsystems store the name of the default SVP. MVS subsystem names are defined within IEFSSNxx members of the SYS1.PARMLIB data set.

The SVP Subsystem

The benefits of running the SVP started task as an MVS subsystem are:

a) Each SVP may perform End-Of-Task processing when a task that has been using its services is removed from the system.

b) Each SVP may perform End-Of-Memory processing when an address space that has been using its services is removed from the system.

c) Each SVP has a memory cell reserved for it (in the SSAT), which it can use to anchor the YSVPARM area that must be associated with each task that is using it. This area contains the information necessary for an SVP to service requests from tasks. The YSVPARM area is built by the SVPLINK SVP request, and is required before any other SVP request may be processed by an SVP.

The SVP subsystem name must be identical to the name of the SVP started task, which is also the SVP name used in the APLSVPxx configuration document, and as an argument to the SVPLINK request.

The SVC Subsystem

The SVP SVC is (usually) an MVS type 3 SVC and each SVP SVC must have access to the name of a default SVP. The code which MVS calls to process a type 3 request should be re-entrant; non-self-modifying code. To avoid tying the SVC permanently to a given SVP, or modifying the code of the SVC processor, it is necessary to store the name of the default SVP for each SVC number outside of the SVC processing code.

Further, a request to an SVP need not include the SVP system number returned by SVPLINK. This is a potential problem because this number is required to index the subsystem data structure (SSAT) in which the YSVPARM area is anchored. If a valid subsystem number is not available, the entire SSAT must be searched, a process which is both inefficient and error-prone.

Both problems are solved by allocating a subsystem to be used by each SVP SVC, purely for storage purposes. This "SVC" subsystem is never active and exists solely for storage purposes. Each SVC processor module (assembly YSVC) has the name of its own unique SVC subsystem assembled into it at installation time.

The SVP information stored in the SVC subsystem is:

a) The default SVP name for users of the SVC. This is placed into the SSCTSUSE field of the SSCT entry, for an SVC subsystem, by a starting SVP. This process is controlled by the SVCSS parameter of the APLSVPxx configuration document.

b) The address of the first YSVPARM created for a task. This address is stored into the SSAT entry belonging to the SVC subsystem (for a given task) to be used for all requests for that task, via the particular SVC, for which SYSTEM= is not specified.

NOTE: Because of use (b), it is advisable to place SVC subsystem entries as closely as possible to the beginning of the IEFSSNxx member of the SYS1.PARMLIB. This is because many non-APL requests to the SVP do not code SYSTEM= and as a result, the subsystem list is frequently scanned in search of these entries. Gains in performance may be achieved by keeping the SVC subsystem entries near the beginning of the IEFSSNxx member.

ALLOCATING A SYSMDUMP DATASET

There is no reliable way to predict the size of a SYSMDUMP dataset for a single site. This is because (1) the DUMP will contain system areas that will vary from site to site, and (2) Shared Variable Processor memory size will vary. A suggested method is to allocate 10 megabytes, obtain some trial dumps, and then choose a size based on the size of the trial dumps.

STARTING, STOPPING, AND CONTROLLING THE SVP

The MVS START (S) COMMAND - The START command creates the MVS address space that the Shared Variable Processor will run in, and specifies the name of the PROCLIB member that contains the JCL for the SVP started task. Since the SVP is an MVS subsystem, the procedure **must** reside in SYS1.PROCLIB. FIGURE 1 is an example of an SVP PROCLIB member. Assuming this is the procedure which initializes the SVP, the START command could be:

```
S SVP0,MEMBER=T1
```

where SVP0 is the name of the PROCLIB member. The MEMBER=T1 keyword override in this example, specifies a particular parmlib member to be used to configure the Shared Variable Processor (Figure 1 shows the default parmlib member to be '00').

The configuration of a Shared Variable Processor is controlled by an APLSVPxx parmlib member (see Figure 2). During initialization, this parmlib member is scanned, and the SVP is configured according to the parameters specified here. These parameters (NAME, CODE, USERS, TRACE, SVCSS, and SIZE) may be overridden by specifying different values in the MVS START command. An example of a START command which overrides Start Parameters specified in the parmlib member is:

```
S SVP0,MEMBER=T1,PRM='USERS=150'
```

which overrides the USERS parameter in parmlib member APLSVPT1. For further information on Start Parameters refer to THE SVP STARTED TASK (SVPS): START PARAMETERS in the CONFIGURATION chapter.

The MVS STOP (P) Command - The STOP command results in an immediate shutdown of the specified Shared Variable Processor. Users of this SVP are not posted or notified, and the next use of the SVP will return the result SVZNA (this is still true if the next request comes after the SVP is restarted). Users must then wait until the SVP is restarted, then issue an SVPLINK and a SIGNON. Programs that do not contain logic to detect and correct this situation may malfunction or ABEND when the SVP returns the unexpected SVZNA response.

The MVS MODIFY (F) Command: CANCEL - The command <F SVPx,CANCEL> is described in the OBTAINING AN SVP DUMP section in this chapter. The MODIFY command results in an abnormal termination of the specified SVP, with an SVP dump if possible.

For further information on the **START**, **STOP**, and **MODIFY** Operator commands, consult the appropriate IBM MVS documentation.

```
//SVP0     PROC  PHASE=SVPS,MBR=00,PRM=''
//*
⍝//*      APE SOURCE - ⍝
//*
//*      RELEASE 19.8 PROC TO RUN SHARP APL SVP SUBSYSTEM '0'.
//*      SHARED VARIABLE PROCESSOR RELEASE 2.0
//*
//*      PHASE  IS THE NAME OF THE PHASE TO RUN
//*      MBR    IS THE LAST TWO CHARACTERS OF THE APLSVPXX
//*             PARMLIB MEMBER NAME
//*      PRM    IS THE SVPS PARAMETERS LIST
//*
//SVP0     EXEC  PGM=∊PHASE,TIME=1440,DPRTY=(14,14),
//               PARM='MEMBER=∊MBR,∊PRM'
//STEPLIB  DD  DSN=SAPL.DIST.LOAD,DISP=SHR
//SYSMDUMP DD  DSN=SVP.SYSMDUMP,DISP=SHR
//SYSPRINT DD  DUMMY
//PARMLIB  DD  DSN=SAPL.DIST.PARMLIB,DISP=SHR
//MSGLIB   DD  DSN=SAPL.R19.MSGLIB,DISP=SHR
```

Figure 1 lists a sample SVP0 proclib member (distributed as APE document 1854339*APL:MVS:CPYSVP0.XMPL.PROC*).

```
/***********************************************************/
/*                                                         */
/* APLSVP00 - SAPL MASTER SVP DEFINITION FOR SUBSYSTEM SVP0*/
/*                                                         */
/*                                                         */
/***********************************************************/
    NAME=SVP0,         /* OFFICIAL NAME OF THIS SVP        */
    CODE=SVP,          /* NAME OF LMOD FOR SVP CSA CODE    */
    SIZE=(262144,FIX), /* LENGTH IN BYTES OF SVSTORE (256K)*/
    USERS=100,         /* NUMBER OF SIMULTANEOUS USERS     */
    TRACE=4096,        /* ALLOCATE A 4K CSA TRACE POOL     */
    SVCSS=S231         /* WE ARE DEFAULT SVP FOR SVC 231   */
```

Figure 2 lists a sample APLSVPxx parmlib member (distributed as APE document 1854339*APL:MVS:APLSVP00.XMPL.PARM*)

OBTAINING AN SVP DUMP

A dump of a Shared Variable Processor may be obtained by one of two methods.

MVS DUMP Command. - The MVS DUMP command may be issued at any time. This generates an unformatted dump into a SYS1.DUMPxx dataset, then permits the Shared Variable Processor to continue processing. To ensure that sufficient information gets into the dump to allow proper analysis, the following dump options (at least) should be specified.

In MVS/SP:

```
        JOBNAME=(SVPx),SDATA=(RGN,NUC,CSA,LPA)
```

In MVS/XA:

```
        JOBNAME=(SVPx),SDATA=(RGN,NUC,CSA,LPA,ALLNUC)
```

SYSMDUMP at SVP ABEND time - When the Shared Variable Processor detects a failure, it requests a dump with dump ranges that include all critical areas of CSA storage. Recommended procedures include specifying a SYSMDUMP DD statement in the SVPx catalogued procedure, which produces an unformatted dump for any SVP failure. Note that MVS processing of the <**CANCEL SVPx,DUMP**> command will **not** provide sufficient information for dump analysis.

This problem may be overcome with a MODIFY command which results in an immediate Shared Variable Processor ABEND: entering the MVS command <**F SVPx,CANCEL**> produces a dump (as long as an appropriate SYSMDUMP statement is present in the SVPx procedure), and removes SVPx from the system. In most cases, all users of the SVP (including APL) will have to be shut down and restarted. The CANCEL Modify command should be used only as a last resort.

INSPECTING AN UNFORMATTED SVP DUMP

The workspace 1854339 *DUMP* may be used to inspect an unformatted dump of a Shared Variable Processor. The dump should be opened normally, specifying 'NONAPL' as the application. This will locate the SVPS linkedit map, and allow other parts of the SVP to be located. There are few facilities in the dump workspace aimed specifically at debugging SVP problems, but most areas of interest may be located with reference to the SVEC area, which can be found via the expression *BASE 'SVEC'*

OPERATIONAL IMPLICATIONS OF THE SVP AS A SUBSYSTEM

Since the Shared Variable Processor is an MVS subsystem, it does not use JES facilities. Some minor implications of this are:

- The SVP may start, during initialization, before JES.
- The SVP may not use SYSIN or SYSOUT data sets.
- The '$HASP395 SVPx ENDED' message, since it is issued by JES, will not be issued for an SVP task.

SVPS CONSOLE MESSAGES

These SVPS console messages are reproduced here for completeness only; the "SHARP APL Messages and Codes" manual (Publication code: 0456 8703 E19) should be consulted as the final authority for message content and meaning. Lower case, boldface text in this table represent values that will be supplied in messages displayed at the console.

APL0050C	REGION SIZE TOO SMALL
APL0113W	INVALID CONTROL INPUT TO SVPS, LAST TEXT SEEN WAS **text**
APL0115W	MEMBER **mbr** NOT FOUND IN PARMLIB
APL0126I	INITIALIZATION COMPLETE
APL0155W	SVSIZE REQUESTED IN **mbr** FOR SVP **<name>** IS TOO SMALL.
APL0159I	STOP COMMAND ACCEPTED
APL0178I	SHARP SVP STATUS
APL0179C	INVALID NAME FOR SVP SUBSYSTEM
APL0192C	ILLEGAL **parameter** CONFIGURATION PARAMETER
APL0193C	ERROR ENCOUNTERED WHILE PARSING **svp** CONFIGURATION
APL0194C	**svp** CONFIGURATION PARAMETER MISSING OR INCOMPLETE
APL0196C	SVP CONFIGURATION PARMLIB MEMBER NOT FOUND
APL0197C	SVP CONFIGURATION PARMLIB MEMBER STACK OVERFLOW
APL0213I	SVP RUNTIME CONFIGURATION ENCOUNTERED A KEYWORD PARAMETER UNKNOWN TO IT
APL0276C	THIS SUBSYSTEM IS NOT IDLE
APL0282C	INITIAL SUBSYSTEM REQUEST RETURNED R15 **hex**, SSOBRETN **hex**
APL0287I	ILLEGAL CALL TO PARMLIB GET ROUTINE.
APL0288I	PERMANENT I/O ERROR ON PARMLIB.
APL0289I	INSUFFICIENT MEMORY FOR PARMLIB I/O.
APL0290I	PARMLIB MEMBER LRECL NOT 80 BYTES.
APL0291C	JOB CANCELLED BY PARMLIB I/O UTILITY.
APL0471C	TOO MANY SVCSS KEYWORDS SPECIFIED
APL0472I	SUBSYSTEM **svcss** NOT FOUND OR ACTIVE

To share and manipulate APL variables, an auxiliary processor must communicate with the Shared Variable Processor using the interface provided for this purpose. The interface to the SVP consists of a collection of COPY code and macros coded in SYSTEM/370 Assembler Language. Programs written in compiled languages can use shared variables by invoking the appropriate assembly language routines via cover/interface routines.

The macros are listed in APE document 1854339*APL:UNION:APOPS..UN*, and the interface definition is in APE document 1854339*APL:UNION:APSVDEFN..UN*. A copy module, 1854339*APL:UNION:SVDEFN..UD*, is included by APSVDEFN; SVDEFN defines the SVP operations and contains all necessary DSECTS.

BACKGROUND INFORMATION

Using the SVP Interface Macros

The macros described in this chapter are intended to provide an Auxiliary Processor with all the facilities required to communicate with another task via the Shared Variable Processor. Most of these macros require arguments, and two control blocks must be defined to pass along user-supplied information. Each macro call returns information on the completion status of the macro as well as any appropriate explicit results. Depending on the requirements of the Auxiliary Processor, SVP requests may be processed one at a time or submitted as chains of requests.

Before a task may set up communication with another task, it must first set up communication with the Shared Variable Processor. A task first issues an SVPLINK to specify one SVP to which all subsequent SVP requests are to be directed. This macro returns the internal SVP number which may be used as a parameter in subsequent requests to reduce the SVP SVC path length. Once the SVPLINK is issued, a task must identify itself to the SVP. This is done by issuing a SIGNON.

When an auxiliary processor has identified itself to the Shared Variable Processor, it may share variables with any other task that is signed on to the SVP. In setting up communication, two tasks must explicitly agree to share variables. Task A makes an offer to share a variable with Task B, and Task B accepts the offer. Both the offer by Task A and the acceptance by Task B are handled by issuing the SHARE macro.

Making an offer is fairly straightforward for the task that initiates communication. This offer may be directed to a specific task or to any task that is signed on to the Shared Variable Processor.

On the other end, a "receiving" task must know that an offer is being made, and more specifically, what offers are being made, before it can accept an offer to share a variable. To this end, an inquiry can be made by issuing the SCAN macro. This macro can be used to inquire about offers from all tasks signed on to the Shared Variable Processor, or to inquire about offers from a specific task. It may be qualified to only report recent offers. With the results of SCAN, a task may match an offer to share a variable, by issuing a corresponding SHARE request.

Once two tasks have made matching offers to share a variable, both tasks may use (reference) the value of this variable, or set a new value. The two partners must synchronize the setting and using of the variable in order to make communication meaningful. This is aided by a state vector associated with each shared variable and two access control vectors associated with each shared variable. In interlocking communication, each partner may access the state vector and access and set the control vector associated with a shared variable.

A set or use of a shared variable is accomplished in two operations. The initial operation obtains exclusive control of the variable to ensure that the partner may not change the value or the status of the variable. Once control of a variable is obtained, a value will usually be set or used. A set or reference of a variable changes the state vector.

To terminate sharing of a single variable, a partner issues a retract for that variable. Once a partner issues a retract for a variable it is no longer shared, and each partner will have a private version. When a partner retracts an offer to share, neither partner can set or reference the other partner's version of that variable.

After Task A has retracted a variable, one of two possibilities may occur; Task A may once again make an offer to share the retracted variable, or Task B may also retract its offer to share the variable. If Task A reoffers, communication may or may not be reestablished. This decision is made at offer time (See discussion of SCVPSX field in the SHARE macro) by Task B. On the other hand, if Task A retracts its offer to share a variable, then Task B may also retract its offer. In this case, communication is terminated, and in order to reestablish communication, both tasks must once again make explicit offers to share a variable.

When a task no longer wishes to share variables, communication with the Shared Variable Processor may be terminated by issuing a SIGNOFF. This tells the SVP that the task no longer requires its services, and allows the SVP to reuse the resources that have been assigned to that task.

Definitions

Degree of Coupling: - The explicit result of an offer to share a variable (⎕*SVO*). The value of this result is the number of processors which have offered to share the variable. The possible values are 0 (not a shared variable), 1 (an offer has been made by one partner but no corresponding offer has been made by the other partner), or 2 (the shared variable is fully coupled, both partners have made offers to share the variable).

PSX: - (Pershare Index) This is an integer alias used by an SVP to uniquely identify a particular shared variable. The PSX is returned in the SCVPSX field of the SCV data area, after a successful SHARE operation. The PSX is an argument to all other SVP operations which refer to a particular shared variable.

Processor ID: - A positive integer to identify the AP. IPSA-written auxiliary processors will have processor IDs in the range 0-901, leaving the range 902-999 available for use by site-written auxiliary processors. IPSA reserves the right to use processor IDs in the range 902-999 for its own products in order to be compatible with auxiliary processors developed in the future by other vendors. For a number of IPSA-written auxiliary processors, the processor ID is a parameter that can be changed by the site to suit its own needs.

Clone ID: - A non-negative integer which qualifies the processor ID. The use of zero may simplify a partner's APL program, since a clone id of zero is assumed when the clone ID is omitted from the argument of ⎕*SVO*.

Chronology: - A positive integer used by the Shared Variable Processor to maintain a chronological order for shared variable offers (SHARE attempts whose degree of coupling is 1). An AP may use the chronology to exclude the consideration of offers earlier than specified, and the SVP resolves the ambiguity of two or more otherwise identical offers by choosing the earliest (i.e. the one with the lowest chronology).

YSVPARM: - This is a Shared Variable Processor generated control block, allocated in fixed (but swappable) LSQA. This area embodies the link between a task and any SVP it may use, and is created either explicitly by the SVPLINK request, or implicitly by the first SIGNON if no SVPLINK was issued. It is used as a parameter passing area, a communication area, and a work area, by the components of the SVP.

SVP Control Blocks

Two user-provided control blocks are required for an auxiliary processor to communicate with the Shared Variable Processor. A Processor Control Vector (PCV) is required when establishing a connection with the SVP and may be used when disconnecting. A Share Control Vector (SCV) is used for inquiries about unaccepted offers to share, to extend an offer to share, and to obtain the degree of coupling of an existing shared variable; figure 3 lists the definitions of both of these control blocks. The control blocks SCV and PCV are defined in the APE document 1854339*APL:UNION:SVDEFN*. These versions may be used as a guide to constructing SVP control blocks.

```
* PROCESSOR CONTROL VECTOR.
*
PCV        DSECT  ,          PROCESSOR CONTROL VECTOR.
PCVID      DS     F          PROCESSOR ID.
PCVNID     DS     F          CLONE ID OF THIS INCARNATION OF
*                            PCVID.
PCVECB     DS     F          ADDRESS OF POST ARGUMENT FOR
*                            SVSIGNAL.
PCVTRAP    DS     F          TRAP ROUTINE ADDRESS (APLSV ONLY).
PCVL       EQU    *-PCV      PROCESSOR CONTROL VECTOR LENGTH.

*
* SHARE CONTROL VECTOR.
*
SCV        DSECT  ,          SHARED VARIABLE CONTROL VECTOR.
SCVID      DS     2F         PROCESSOR, CLONE ID.
SCVNID     EQU    SCVID+4    CLONE ID.
SCVALUE    DS     A          ADDRESS OF VALUE DESCRIPTOR.
SCVNO      DS     F          SHARED VARIABLE CHRONOLOGY.
SCVPSX     DS     F          SVP ENCODING OF SHARED VARIABLE
*                            NAME.
SCVOTHER   DS     2F         PROCESSOR, CLONE ID OF PARTNER.
SCVONID    EQU    SCVOTHER+4 PARTNER'S CLONE ID.
SCVNAMEL   DS     X          LENGTH OF SURROGATE NAME.
SCVNAME    DS     0C         FIRST CHARACTER OF SURROGATE
*                            NAME
```

Figure 3 SSVP Control Block Definitions

SVP Operating Modes

The SHARP APL Shared Variable Processor is an MVS subsystem that is usually invoked as a subroutine of the host operating system's Supervisor Call processing routines. The SVP operates in supervisor state and can access both its own storage and that of the requesting program. It is serially re-entrant: each request must be completed before a subsequent request can be begun (this serialization is accomplished in MVS by the SVP using the local lock of the SVP address space).

Typically, a Shared Variable Processor call includes a request code and arguments appropriate to the request. If the request is valid, it is processed and explicit (values returned in general registers) and implicit (storage addressed by an argument) results are returned. This mode, in which a single request is passed and processed, is called storage mode.

In addition to a set of storage mode requests, the Shared Variable Processor provides two requests, SSVPCC and SSVPCX, that are roughly equivalent to a START SUBCHANNEL instruction. SSVPCC takes as its argument a chain of commands resembling 8 byte CCWs. These CCWs do not allow for 31-bit addressing. SSVPCX takes as as its argument a chain of 12 byte extended CCWs (APE reference 1854339:APL:UNION:CCWX.IPSA.UM) which permit 31-bit addressing.

The argument is the address of a chain of one or more Channel Command Words whose command fields contain valid SVP request codes and whose address and length fields describe arguments appropriate to the requests (see 1854339*APL:UNION:SVDEFN* for details). The explicit result of SSVPCC or SSVPCX returned in registers zero and one is an appropriate Channel Status Word. This mode, in which a chain of commands are passed and processed, is called Channel Mode.

Note: Where explicit results of each Shared Variable Processor request are required, storage mode is preferable, since these are not obtained in channel mode. Where explicit results are not required, channel mode is preferable, since the number of supervisor calls required may be reduced.

Calling Macros

The Shared Variable Processor macros are provided in APE document 1854339*APL:UNION:APOPS*. The general format of the macro call is:

label MACRO arguments

Such a call first loads registers as follows:

GPR 15 - An SVP command code in bytes 2 and 3, optionally preceded by an SVP number in bytes 0 and 1 of the register. Note that a (small) performance gain can be realized by specifying the SVP number on all SVP requests. The system SVP number is specified by coding the SYSTEM= parameter in individual macro calls.

GPR 0,1 - Arguments relevant to the command code.

The macro then executes a supervisor call instruction whose SVC code is established by the linkage editor. Control is always returned to the instruction immediately following the SVC with all registers unchanged, except for:

GPR 15 - SVP Return code in bytes 2 and 3. Bytes 0 and 1 will be either 0 or an explicit halfword result.

GPR 0,1 - Explicit results, if any.

In addition, storage addressed by one or more of the arguments might have been altered by the SVP.

Any macro can be labelled, and the label is assigned to the address at the first generated instruction. Arguments are positional and can be omitted to indicate that the appropriate registers are already loaded. An argument can consist of a number enclosed in parentheses to indicate an argument in a general register, or it can be a symbolic address for which System/370 addressability has been established, or in the case of ACV it can be a self-defining term that evaluates to the desired access control vector.

Return Codes

Return codes are stored in the low-order halfword of GP15 at the completion of a Shared Variable Processor request. The value of the return code indicates the status of the request.

Since the value of return codes may be release-dependent, in the following sections, all return codes are referred to by labels. The definitions of all return codes, along with their respective labels, are contained in SVDEFN.

A "normal end" return, SVZNE, indicates that the request was accepted and executed, and the explicit results contained in GP0 and GP1, as well as any implicit results, can be examined for further information.

A return smaller in magnitude than SVZNE indicates that the argument was accepted but not completed because of access restrictions, temporary storage limitations, or the command was intentionally incomplete. A return greater than SVZNE indicates that the command was rejected because it was invalid, impossible to complete, or because of a major shared variable storage resource limitation. The parity of the return code is used to indicate the degee of coupling at the completion of any command addressed to a specific shared variable. If the degree of coupling is 1, then SVZNOS (EQU 1) is added to the return code, making it odd; an even return code indicates a degree of coupling of 2.

The following return codes can be returned for any request and, with the exception of SVZNA, indicate an error in the AP,

SVZNA:	SVP is not available.
SVZARG:	Argument error; an argument failed an SVP validity check.
SVZSPE:	Storage Protect Exception; an argument specifies an address that is outside of the AP region or partition.
SVZNSO:	Not Signed On; returned for all requests except signon if the issuing task has not properly signed on to the SVP (or has signed off).

Posting

When issuing a SIGNON, a task may designate the address of a word (ECB) to be posted by the Shared Variable Processor signal routine. Anytime after the SIGNON, the task may issue an MVS WAIT macro, including this ECB in its ECB list, whenever it must wait for an SVP-related condition to end (e.g. temporary storage shortage, access control interlock). This ECB will be posted whenever the SVP detects an event that may affect this task.

MANAGING A CONNECTION TO THE SVP

Establishing the SVP Link: label SVPLINK name

SVPLINK is used prior to the SIGNON command to specify the name of the SVP to which all subsequent SVP operations are to be directed. Programs which do not use SVPLINK (do not specify a particular SVP) are automatically linked to the SVP which is the default for the SVC they issue. SVPLINK also returns the internal SVP number (the SVP's subsystem index value) assigned to the named SVP; this number is an optional argument to all other SVP operations. This optional parameter (the keyword 'SYSTEM') need only be supplied if concurrent links are established to multiple SVPs.

Note that a small reduction in SVP SVC path length can be achieved by specifying the SYSTEM number on all SVP macros: failure to specify the SYSTEM number results in an extra access to the SVC subsystem data structure.

SVPLINK Return codes:

SVZNE AP is successfully linked.

SVPLINK Results: (if low-order halfword of R15 = SVZNE)

GPR15 High-order halfword: the SVP number of the SVP to which the link was made.

GPR0 The address of the <YSVPARM> area created by the link. The YSVPARM area is a fixed (LSQA) area used as a formal parameter area, and a communications area used by the SVP SVC handler.

GPR1 The address of an information area within the SVP SVC code. This should be used only by SHARP-coded routines, as it may change without notice.

SVP Linkage Query: label YCOMAD name

YCOMAD allows a program to determine certain things about the SVC it is using to communicate with the SVP. Most programs will have no need to issue YCOMAD. WARNING: YCOMAD results may change from release to release of the Shared Variable Processor.

YCOMAD return codes:

SVZNE: the output registers have been set

YCOMAD results:

R0 — 0 if the SVC subsystem associated with this SVC has not been defined to MVS.

(or) — Address of a halfword of zero if the SVC subsystem of this SVC has not been mentioned in the SVCSS parameter of an SVP.

(or) — Address of the four-character EBCDIC name of the default SVP for this SVC.

R1 — The address of an internal address vector within the YSVC assembly. This is for debugging use only.

Establishing the Connection: label SIGNON pcv[,SYSTEM=svp-number]

NOTE: If a SIGNON is issued by a task that has not issued an SVPLINK, an SVPLINK is automatically attempted with a default Shared Variable Processor name. The default name will be the SVP which has most recently been started with a parmlib member containing an SVCSS parameter referencing the SVC subsystem associated with the SVP SVC used to request the SIGNON operation.

When this macro is called, the command fields of the processor control vector (PCV) must contain the following information:

PCVID,PCVNID — Processor identification, clone ID.

PCVECB — Address of a word (ECB) to be posted by the SVP signal routine.

PCVTRAP — Zero (non-zero indicates APLSV compatibility mode).

SIGNON Return codes:

SVZNE	AP is successfully connected.
SVZNIU	The specified Processor ID, Clone ID are already in use by another processor.
SVZASO	The specified Processor ID, Clone ID are already in use, but belong to this AP.
SVZPPF	SVP processor table is full.

SIGNON Results: none.

Disconnecting from the SVP: label SIGNOFF pcv[,SYSTEM=svp-number]

The argument is either the address of a processor control vector previously used as an argument to SIGNON, in which case the specified identification is disconnected, or it can be zero, indicating that all identifications belonging to the issuing task are to be disconnected. SIGNOFF implies the retraction of all shared variables belonging to the disconnecting routine.

SIGNOFF Return codes:

SVZNE	Processor is now signed off.
SVZNSO	Not Signed On. The processor specified in the processor control vector was not connected to the SVP.

SIGNOFF Results: none.

MANAGING SHARED VARIABLES

Inquiring for Offers: label SCAN scv[,SYSTEM=svp-number]

An AP can determine whether any variables that meet certain criteria have been offered to it by "scanning" for offers.

When this macro is called the command fields of the share control vector (SCV) must contain the following information:

SCVID,SCVNID
Processor ID, Clone ID that were used in the processor control vector when the SIGNON was issued.

SCVNO
Contains the lower bound of chronologies to be considered. Only those offers with a chronology greater than SCVNO will be examined by the SVP; all offers can be examined by using an SCVNO of zero.

SCVPSX
If negative, signifies that variables that have previously been shared with the inquiring processor and subsequently retracted are not acceptable offers to the processor. This permits a processor to retract shares "permanently".

SCVOTHER,SCVONID
Zero, indicating that offers from any processors are to be considered, **OR** a Processor ID, Clone ID, indicating that only offers from a specific partner are to be considered.

SCVNAMEL
A non-zero value indicating the longest variable name to be considered, **OR** the number of characters in the name which begins at SCVNAME.

SCVNAME
Binary zero, indicating that all names are to be examined **OR** non-zero, indicating that only those variables which have the specified name are to be considered.

SCAN Return codes:

SVZNOF
No Offer Found. There are no offers that match the specifications which are contained in the share control vector; the share control vector is unchanged.

SVZNE
A matching offer was located, the share control vector has been altered.

SCAN Results (SVZNE):

The share control vector is altered as follows:

SCVNO The chronology of the located offer.

SCVPSX The SVP internal name for the variable.

SCVOTHER,SCVONID
Processor-clone ID of the offering AP.

SCVNAMEL,SCVNAME
Length and name of the offered variable. If SCAN was invoked with the first character of SCVNAME non-zero, these fields are unchanged. If the first character was zero, SCVNAMEL has been changed to the length of the name which was found. SCVNAME contains the name unless the name is longer than the value of SCVNAMEL when SCAN was invoked, in which case the name has been truncated to this limit.

Offering to Share a Variable: label SHARE scv[,SYSTEM=svp-number]

Except in the cases where SCVNAME has been truncated or SCVALUE has not been properly initialized, the share control vector that is completed by SCAN is a proper argument for SHARE and can be used to accept the offer just located. The resultant degree-of-coupling must be examined, however, in case the offering processor has retracted the offer during the interval between the SCAN and the subsequent SHARE. Otherwise, when this macro is called, the command fields of the share control vector must contain the following information:

SCVID,SCVNID
Processor ID, Clone ID that were used in the Processor Control Vector when the SIGNON was issued.

SCVALUE Negative one (¯1) indicating no value, OR the address of the descriptor that describes a possible initial value.

SCVPSX Bytes 1 - 3 may contain an internal name from a previous scan, in which case the specified variable must otherwise match the share control vector, OR may be zero indicating that offers are to be searched for one which is acceptable. If bytes 1 - 3 contain an internal name and the sign bit (leftmost bit) of byte 0 is set the offer will only be accepted if it was not previously shared and retracted. That is, if the Sign bit is 1 in this field, the offer can only be matched once. If the partner retracts and then re-offers, the re-offer will not be matched until this side issues a retract and a subsequent re-offer.

SCVOTHER,SCVONID
Zero, indicating that offers from any processors are to be considered, OR a Processor ID, Clone ID, indicating that only offers from a specific partner are to be considered.

SCVNAMEL,SCVNAME
Must denote a valid variable name.

SHARE Return codes:

SVZNE The offer to share has been processed (but not necessarily matched: see GPR0).

SVZPSF SVP Shared Variable Table is full.

SHARE Results (SVZNE):

EXPLICIT: GPR0 contains the degree-of-coupling of the offer just made. GPR0 = 0 indicates that a share control vector which specified a non-zero SCVPSX did not match the variable which has the specified internal name; the share control vector has not been altered. GPR0 = 1 indicates an offer was extended, GPR0 = 2 indicates that an offer was accepted.

SHARE executed with a share control vector that describes a variable which has already been offered is an inquiry for the degree-of-coupling.

SVZNOS is not set in the return code to indicate a degree of coupling of one. GPR0 must be examined to test for this.

IMPLICIT: SCVNO contains the chronology of an offer that was extended or accepted.

When SCVOTHER,SCVONID were zero when SHARE was invoked, and when the degree-of-coupling is two, they contain the identification of the processor with which the variable is shared.

Retracting an Offer to Share: label RETRACT psx[,SYSTEM=svp-number]

RETRACT Return Code:

SVZNE	Offer has been retracted.

RETRACT Results: (SVZNE)

EXPLICIT: GPR0 contains the degree-of-coupling before retraction.

IMPLICIT: If the degree-of-coupling was one (1), the offer has been erased and the shared variable no longer exists. If two (2), the shared variable may remain as an offer to the retracting task.

ACCESS CONTROL

A four-element Access State Vector and two four-element Access Control Vectors are associated with each shared variable. These are described in the "SHARP APL Reference Manual" (Publication code: 79RM05). The auxiliary processor's contribution to the access control can be set, and both the access state and the combined access control vector can be examined.

Set Access Control: label ACV psx,cv[,SYSTEM=svp-number]

(SVP operation code SVPSVC)

Psx is a valid SVP internal name returned in SCVPSX at the conclusion of a successful SHARE. Cv is the desired access control vector, a binary value between zero and 15 inclusive.

ACV Return codes: SVZNE (& SVZNOS)

ACV Results (SVZNE):

EXPLICIT: The low order byte of GPR 0 contains the current access state in the first hexadecimal digit; the second digit contains the access control vector which is determined by combining the value just set with the partner's contribution. (the high-order bytes of GPR0 are undefined after this request)

IMPLICIT: None.

Access State/Control Inquiry: label ACV psx[,SYSTEM=svp-number]
(SVP operation code SVPSVS)

ACV Results: As for ACV which also sets access control (above).

Note on channel mode usage:

The address in the channel mode form of ACV inquiry (SVP operation code SVPSVS) points at a list of PSX values. Each has a 24-bit PSX value right justified in a 32-bit fullword. Values of zero (0) are ignored. After the call, each fullword will contain (in the leftmost byte) the 8 bits of information returned in GPR0 in storage mode. This provides a relatively fast way of obtaining the status of many shared variables at one time.

SHARED VARIABLE DATA TRANSFER OPERATIONS

A shared variable is used (read) or set (written) by a sequence of two or more operations. The first is an initial selection that obtains exclusive control of the variable so that the partner is excluded from all data transfer operations and which returns information about the current status of the variable so that scheduling decisions, storage allocation, etc., can be performed while the access state remains unchanged. Operations that actually transfer data are only accepted when the variable is under exclusive control obtained by an appropriate initial selection.

Initial Selection Operations:

The initial selection operations are as follows:

Use initial selection:	label USEINIT psx[,SYSTEM=svp-number]
Set initial selection:	label SETINIT psx[,SYSTEM=svp-number]
Obtain exclusive control:	label SEIZE psx[,SYSTEM=svp-number]

USEINIT, SETINIT, and SEIZE Return codes (& SVZNOS):

SVZLOCK	the variable is inaccessible, either because it is under the partner's exclusive control, or because the access control vector inhibits the attempted operation in the current access state.
SVZNE	Initial selection completed; the variable is now under exclusive control.
SVZIVS	Invalid sequence: attempting to obtain control of a variable already controlled.
SVZNV	(USEINIT only) Access control does not inhibit a use of the variable, but SVP has no value for it in Shared Variable Storage.

USEINIT, SETINIT, and SEIZE Results (SVZNE):

EXPLICIT: GPR0 contains the length in bytes of the current shared variable value, or contains ¯1 to indicate there is no value present in SVS.

Data Transfer Operations:

(In the following, "value" denotes the address of a double-word descriptor; the first word contains the address of the value and the second contains its length. **Note:** for APL values, this length is eight bytes shorter than the value contained in SBCNT; the data begins at SBTYP.)

Use transfer operation:

label USEDATA psx,value[,SYSTEM=svp-number]

USEDATA Return codes (& SVZNOS):

SVZNE Value transferred from SVS, access state changed as appropriate, control released.

SVZIVS The variable denoted by SCVPSX was not under exclusive control from a previous USEINIT.

Set transfer selection:

label SETDATA psx,value[,SYSTEM=svp-number]

SETDATA Return codes (& SVZNOS):

SVZSVSF Shared Variable Storage temporarily full, data not transferred, control retained.

SVZNE Value transferred to Shared Variable Storage, access state changed as appropriate, control released.

SVZIVS The variable was not under control obtained by a SETINIT.

SVZVTL Value larger than the maximum acceptable by SVP, no data transferred, control released.

Copy transfer selection:

label COPYDATA psx,value[,SYSTEM=svp-number]

Note: COPYDATA is intended as an intermediate step in sequences such as are required for APL indexed specification. It does not release control and does not change the access state. It may follow any initial selection operation which obtains control.

COPYDATA Return codes (& SVZNOS):

SVZNE Value has been transferred from SVS.

SVZIVS Variable is not under exclusive control, or an attempt was made to copy a non-existent value from SVS after receiving an explicit result of ¯1 from the initial selection.

Release transfer selection:

label RELEASE psx[,SYSTEM=svp-number]

Used to release control of a variable when control has been obtained but when the data transfer is not desired.

CHANNEL MODE OPERATION

The use of the macros described above, where each operation is requested with a separate call to the Shared Variable Processor, is called Storage Mode. An alternative to Storage Mode is Channel Mode. In Channel Mode operations, a sequence of commands is constructed, and passed to the SVP in a single request. These requests are represented in a manner similar to a chain of CCW's (IBM 370 Channel Command Words), with Shared Variable Processor operation codes replacing the channel/device operation codes. In this mode of operation, the SVP emulates the processing performed by the 370 channel processor, including command decoding, TIC (Transfer In Channel) processing, chaining and other flag-controlled processing, and CSW (Channel Status Word) creation at chain-end.

Since the original "FORMAT-0" CCW does not support 31-bit addressing, the introduction of MVS/XA required a CCW format with an expanded address field. The 370/XA "FORMAT-1" CCW solves this problem by using a byte unused in the "FORMAT-0" CCW: unfortunately, the Shared Variable Processor CCW format had a use defined for this byte. Accordingly, an extended "FORMAT-X" CCW was designed solely for 31-bit SVP usage. As in the 370/XA architecture, the two formats may never be mixed in a single chain. The SSVPCC macro is used to pass a "FORMAT-0" chain to the SVP and the SSVPCX macro is used for a "FORMAT-X" chain.

Notes on Channel Mode Operation

1) Channel mode can significantly decrease the cost of using the Shared Variable Processor, since the fixed overhead associated with a request can be shared among an entire sequence of operations. Note, however, that the results normally returned in R0 and R1 by storage mode requests are not available to a channel mode caller.

2) Since the SCVPSX is passed in the request rather than in the CCW chain, all operations in a given chain will refer to a single shared variable.

3) Channel mode provides the only way to obtain (usually via COPYDATA) part of a value, since SVZILI will end a Channel Mode request after the requested part of the data is transferred.

The Transfer In Channel (TIC) operation is defined for Shared Variable Processor CCW chains, allowing a single chain to be constructed from non-adjacent CCWs.

FORMAT-0 CCW Chain:

label SSVPCC psx, first-ccw[,SYSTEM=svp-number]
.0SSVPCC passes a chain of FORMAT-0 CCW's to the Shared Variable Processor for interpretation. Each CCW has the following format:

Byte 0	Op (Command Code)
Bytes 1 - 3	Address
Byte 4	Flags
Bytes 5 - 7	Count

Op - The Shared Variable Processor command code, such as SVPUDT, which is placed into byte 3 of General Register 15 during a storage-mode SVP call. Command codes are defined in APE document 1854339*APL:UNION:SVDEFN*.

Address - The address of the associated data area: the PCV for SIGNON, SIGNOFF, the SCV for SHARE or SCAN, or the data area for USEDATA, COPYDATA, or SETDATA.

Flags - X'80' (Data Chain): Causes the area described by the Address/Count fields of the next CCW to be considered an extension of the area described by this CCW.

X'40' (Command Chain): Causes the CCW immediately beyond the current CCW to be executed if the operation described by this CCW is successfully completed.

X'20' (Suppress Incorrect Length Indicator): Undefined, must be 0.

X'10' (SKIP Data Transfer): Suppresses data transfer for the current CCW's area. Valid only for USEDATA/COPYDATA.

X'08' (Program Controlled Interrupt): Not used by SVP, must be 0.

X'04' (Indirect Address): Undefined, must be 0.

X'02' (Suspend): Undefined, must be 0.

X'01' Undefined, must be 0.

Count The length of the area referred to by the current CCW, in bytes.

The Assembler Pseudo-opcode CCW may be used to aid in the construction of FORMAT-0 Shared Variable Processor command chains. Such a chain will be interpreted until an operation experiences an abnormal end, or until a CCW without a chaining bit is processed. When this happens, the channel mode operation is considered complete and a "Channel Mode Status Word" (CSW) is returned to the caller of SSVPCC in R0 and R1.

The CSW returned in R0 and R1 has the following format:

Bytes 0 - 3	Address
Bytes 4 - 5	Flags
Bytes 6 - 7	Count

Address The address of the CCW that would have been fetched if processing had continued. This is 8 bytes beyond the address of the last CCW that was processed (in FORMAT-0

	Mode).
Flags	These are set to loosely correspond to the flags set by the System 370 channel in a CSW; they are not normally used by a caller of the SVP.
Count	The number of bytes of the area described by the last-processed CCW which were not processed (either not read by the SVP or not filled by the Shared Variable Processor). This is often called the residual count.

The return code set into R15 by the Shared Variable Processor has the storage-mode meaning for the last-processed CCW.

FORMAT-X CCW Chain:

label SSVPC psx, first-ccw [,SYSTEM=svp-number]

SSVPCX passes a chain of FORMAT-X CCW's to the Shared Variable Processor for interpretation. Each CCW has the following format:

Byte 0	Op (Command Code)
Bytes 1 - 3	(Unused)
Byte 4	Flags
Bytes 5 - 7	Count
Bytes 8 - 11	Address

Although the displacements are slightly different, the fields are defined exactly as in a FORMAT-0 CCW (See SSVPCC).

The effects and results of SSVPCX are identical to those of SSVPCC, with the following exceptions:

- Since CCW's are twelve bytes in length, the "next CCW" pointer is advanced by 12 bytes each time, and the Address returned in the CSW is thus 12 bytes beyond the last CCW processed, rather than 8.
- The IPSA-supplied macro, CCWX, can be used rather than the CCW pseudo-operation, to generate FORMAT-X CCW's.

COMPONENTS OF THE SHARED VARIABLE PROCESSOR

The SHARP APL Shared Variable Processor consists of a number of program segments that interact directly with the host operating system.

SITELCL

(APE reference 1854339*APL:UNION:SITELCL.IPSL.UM*)

The SITELCL macro is the source of all site configuration information required during Assembly and Linkedit of Sharp APL components. The SITELCL macro defines

- The APL SVC number (default = 254). This SVC should be reserved for APL usage, but no MVS system changes are required to support it. APL uses SVC screening to intercept invocations of this SVC, and all processing is done within APL (Installations running an SVP without APL should specify the default value, although in this case the APL SVC will never be issued).
- The SVP SVC number (default = 231) and type (T2, T3, or T4). The SVP normally uses a type 3 or 4 SVC. This is installed into the SYS1.LPALIB system dataset, or a LNKLSTxx concatenation (see "YSVC" below).
- The SVC subsystem name. Each SVP SVC must store a default Shared Variable Processor name, as well as certain information for each MVS task using the SVP. To accomplish this, an MVS subsystem name is reserved for each SHARP SVP SVC number in the MVS system (see MVS Subsystem Structure). This subsystem is called the SVC subsystem for the associated SVC number. A recommended convention is to give subsystems names of the form Snnn, where "nnn" is the number of the associated SVC (i.e. S231 would be the SVC subsystem associated with SVC 231). There are no actual restrictions on the name other than the MVS restriction that subsystem names be no longer than four characters. All SVC subsystems must be defined to the MVS system in an IEFSSNxx document. Since the SVP SVC processor frequently scans the queue of MVS subsystem names, it is advisable to place any SVC subsystem names near the start of the IEFSSNxx document.

SVP

(APE reference 1854339*APL:MVS:LKYSVP.IPSA.LK*)

This module, which resides in the APL LOAD Library, is the Shared Variable Processor global code; it processes all shared variable requests. The name of this load module is specified in the "CODE=" parameter of the SVPS APLSVPnn parmlib member, or in the "CODE=" parameter of the SVPS EXEC parm.

YSVC

(APE reference 1854339*APL:MVS:YSVC*)

This module is normally added to the SYS1.LPALIB dataset or a concatenation, and processes type 3 or 4 SVC SVP requests. Type 2 processing is supported for compatibility with previous releases, but is not recommended since linking code into the MVS nucleus should be avoided whenever possible.

YSVC is a small module, and improved performance and flexibility can be obtained by moving it into the FLPA (i.e. naming it in an IEAFIXxx member of SYS1.PARMLIB). By preference, FLPA modules reside in members of the LNKLSTxx concatenation. This is a site option and the Shared Variable Processor will function properly if YSVC is simply added to SYS1.LPALIB.

NOTE: The naming conventions for Type 3 and 4 SVC LOAD MODULES are as follows: the Load Module name must be in the form IGC00nnp where "nnp" is the EBCDIC/PACKED form of the SVC number. Thus 231 is represented as 23A; 240 is represented as 24⌿ where ⌿ is the EBCDIC code for "plus zero", (left brace, EBCDIC X'CO'). This character can be generated in APE with the COLUMN COMPRESS character, ⌿. The Assembler does not recognize this character as part of a valid symbol name.

SVSTORE

(APE reference 1854339*APL:UNION:SVSTORE.UC*)

This is an area of storage configured in CSA, that consists of: a fixed area (604 bytes); a Shared Variable Processor locals area (1024 bytes); space for the processor table entries, PERPROCs (60 bytes×1+number of processors); and space for shared variable data and share table entries, PERSHAREs. This last area is dynamically shared between PERSHAREs and data, and is not specifically allocated to either: it is formed from that area of SVSTORE left when the other three areas have been allocated. SVSTORE is defined by the "SIZE=" parameter of the SVPS configuration member. The address of SVSTORE is contained in the SVP subsystem SSVT as well as in the SVEC area within SVPS. In MVS/XA, ECSA (Extended Common Storage Area) is used for SVSTORE. This provides CSA storage relief for MVS/XA sites.

SVPS

(APE Reference 54339*APL:MVS:SVPS.IPSA*)

This is the executable code for the Shared Variable Processor started task. Most of the SVP processing during normal SVP operations takes place within the global (CSA) code (load module "SVP"), but all initialization (including loading the global code), termination, and MODIFY command processing is performed by SVPS.

Initialization is accomplished under the control of a configuration document, normally stored as a member of SYS1.PARMLIB. Initialization overrides may be provided in the SVPS EXEC parameter.

THE SVP STARTED TASK (SVPS): START PARAMETERS

Each APLSVPxx parmlib member controls the configuration of a single Shared Variable Processor subsystem. Figure 2 lists a sample APLSVPxx parmlib member.

(distributed as APE document 1854339*APL:MVS:APLSVP*00.*XMPL.PARM*)

Parameter: NAME=name
maximum name length: four characters
Parameter required: Yes.

The NAME= parameter specifies the name of the Shared Variable Processor, which must agree with the name of the subsystem catalogued Procedure (Fig. 1). The name must be defined to the MVS system as a "subsystem", via an IEFSSNxx member in SYS1.PARMLIB.

Parameter: CODE=name
maximum name length: eight characters
Parameter required: Yes.

The CODE= parameter specifies the name of the Shared Variable Processor global (CSA) load module. This module is loaded into CSA from the APL Steplib during SVP initialization.

Parameter: USERS=value
value range: 1 to ¯1+2*31
Parameter required: Yes.

The USERS parameter specifies the maximum number of unique processors (PERPROCS) that can be signed on to the Shared Variable Processor at a given time.

A number of tasks qualify as processors; some examples follow. Each APL task using the Shared Variable Processor is a processor, as is each of the SHARP APL auxiliary processors. For most of the APs, each use of the AP by an APL user is also a processor. Each terminal session signed on to the APL system through IDSH may appear as four processors, corresponding to AP2, AP6, AP124, and AP126. Exceptions to this are: IMVS, which usually appears as two or three processors, and CONH, each copy of which appears as a single processor.

Values of the USERS parameter lower than 40 may cause problems in starting APL. While this is a site-dependent parameter, the minimum recommended value is 100.

Parameter: TRACE=value
value range: 0 to 1048576
Parameter required: Yes.

The TRACE parameter specifies the size, in bytes, of the Shared Variable Processor trace table. If 0 is specified, tracing is disabled. Otherwise, the value is rounded up to a 2K multiple. The SVP trace table (and the TCA which controls its use) is allocated in CSA (Extended CSA in MVS/XA).

Parameter: SVCSS=name
Maximum name length: four characters
Parameter required: No
Parameter specified a maximum of 16 times.

The SVCSS parameter sets up a default Shared Variable Processor for a particular SVC number. Each SAPL SVP SVC must be associated with a (unique) SVC subsystem (see 'SVC Subsystem' under the description of the SITELCL macro). When an application issues a SIGNON without having issued an SVPLINK first, the SVC code uses the default value found in the associated 'SVC subsystem' SSCT to determine the SVP to be used. An SVC subsystem is never active, but must be defined to the system (via IEFSSNxx) as a normal subsystem. For performance reasons, SVP 'SVC subsystem' entries should be defined near the beginning of the IEFSSNxx document.

Parameter: SIZE=(value,fix)
value range: value; 10K to ¯1+2*31
fix; set to 'FIX' or 'NOFIX'
Parameter required: Yes.

The SIZE parameter specifies the size, in bytes, of SVSTORE. The following formula may be used to calculate the approximate size of SVSTORE; USERS is the value assigned to the USERS parameter, and VARSIZE is the largest shared variable (measured in bytes) that is to be passed through the SVP:

SIZE=2000+((1+USERS)×60)+(2×VARSIZE)

Fix is normally set to 'FIX'. Setting Fix to "NOFIX" will result in the allocation of Shared Variable Processor memory from non-pagefixed CSA. This may be preferable in situations with small user populations, where SVP performance is not a critical concern. At most sites, the SVP is a critical performance resource, and the use of fixed CSA memory is preferable. For further information, refer to the description of SVSTORE in the COMPONENTS OF THE SHARED VARIABLE PROCESSOR section of this chapter.

The naming conventions and optimization tips included here, are all mentioned under other topics in this manual. These are grouped together here for convenience only.

The Shared Variable Processor subsystem name, the SVP started task, the SVP name used in the APLSVPxx configuration document, and the SVP name used as an argument to the SVPLINK request, must all be identical.

It is advisable to place SVC subsystem entries as close as possible to the beginning of the IEFSSNxx member of SYS1.PARMLIB. Many non-APL requests to the Shared Variable Processor, choose not to code SYSTEM=. As a result, the subsystem list is frequently scanned in search of these entries. Normally this list is quite short, but it is possible that small gains in performance may be realized by keeping the SVC subsystem entries near the beginning.

An SVC (default = 254) should be reserved for APL usage. No MVS system changes are required to support this SVC, as SVC Screening intercepts all invocations of this SVC.

A recommended convention is to give the SVC subsystems names of the form Snnn, where "nnn" is the number of the associated SVC (i.e. S231 would be the SVC subsystem associated with SVC 231).

Improved performance and flexibility can be obtained by moving the YSVC module into the FLPA (i.e. naming it in an IEAFIXxx member of SYS1.PARMLIB). By preference, FLPA modules reside in members of the LNKLSTxx concatenation. This is a site option and the Shared Variable Processor will function properly if YSVC is simply added to SYS1.LPALIB.